A FLEXIBLE FRAMEWORK FOR COLLABORATIVE

VISUALIZATION APPLICATIONS USING JAVASPACES

THESIS

Sean C. Butler, Second Lieutenant, USAF

AFIT/GCE/ENG/01M-01

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

A FLEXIBLE FRAMEWORK FOR COLLABORATIVE

VISUALIZATION APPLICATIONS USING JAVASPACES

THESIS
Sean C. Butler, B.S.
Second Lieutenant, USAF
AFIT/GCE/ENG/01M-01

AFIT/GCE/ENG/01M-01

A FLEXIBLE FRAMEWORK FOR THE DEVELOPMENT OF COLLABORATIVE

VISUALIZATION APPLICATIONS IN JAVASPACES

THESIS

Presented to the Faculty of the

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

of the Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the Degree of Master of Science in

Computer Engineering
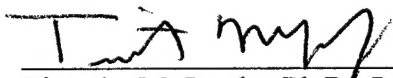
Sean C. Butler, B.S.

Second Lieutenant, USAF

March 2001

AFIT/GCE/ENG/01M-01

A FLEXIBLE FRAMEWORK FOR COLLABORATIVE

VISUALIZATION APPLICATIONS USING JAVASPACES

Sean C. Butler
Second Lieutenant, USAF

Approved:

_____   5 Mar 01
Timothy M. Jacobs, Ph.D., LtCol, USAF (Chairman)   date

_____   5 Mar 01
Scott A. Deloach, Ph.D., Maj, USAF (Reader)   date

_____   5 MAR 01
Karl S. Mathias, Ph.D., Maj, USAF (Reader)   date

Acknowledgements

I would like to express my sincere appreciation to my faculty advisor, Lt. Col. Timothy Jacobs, for his guidance and support throughout the course of this thesis effort. I would also like to thank Dr. Douglas Holzhauer and Capt. Robert Duncomb of the Air Force Research Laboratory, Information Directorate, for all of their invaluable time spent making this research possible.

Sean C. Butler

Table of Contents

List of Figures

List of Tables

AFIT/GCE/ENG/01M-01

Abstract

The complexity of modern tasks is rising along with the level of technology. Two techniques commonly used to deal with complexity are collaboration and information visualization. Recently, computer networks have arisen as a powerful means of collaboration, and many new technologies are being developed to better utilize them. Among the newer, more promising of these technologies is Sun Microsystems' JavaSpaces™, a high-level network programming API. This thesis describes a tool for developing collaborative visualization software using JavaSpaces—an application framework and accompanying toolkit.

In addition to a detailed description of the framework, the thesis also describes an application implemented using the framework, discusses the benefits of development under the framework, evaluates the performance of JavaSpaces in the context of the framework, and addresses the issue of network bandwidth limitations, which are a concern when developing visualizations that deal with large data sets.

A FLEXIBLE FRAMEWORK FOR COLLABORATIVE

VISUALIZATION APPLICATIONS USING JAVASPACES

Chapter 1 - Introduction

1.1 Overview

As the state of technology advances, there is an ever-increasing level of complexity inherent in the products utilizing that technology, be it software, architecture, or even the planning and coordination of a modern military mission. The amount of data associated with this complexity is rapidly becoming correspondingly vast. One of the primary tools our civilization has developed to deal with dauntingly complex projects is collaboration, the practice of multiple individuals working in concert to achieve a common goal. Another tool is information visualization, the study of how best to represent data in a visual form to aid in comprehension.

1.1.1 Collaboration

While collaboration has been utilized to one degree or another since before recorded history, relatively recently we have focused on improving the efficiency of working together. With the advent of widespread computer networks and the Internet, a new, powerful tool for collaboration has been made available to us, and only now are we making significant strides towards finding the optimal means of utilizing it.

The Java™ programming language is a relatively new development with remarkable potential for exploiting computer networks. The primary appeal of Java is, without question, its ability to run on any platform that has the appropriate virtual machine. This property makes Java the ideal language for network programming, as the same software can be used on every machine on a heterogeneous network. As a result, there have been several forays into developing Java-based network and collaboration APIs, ranging from relatively low-level communication protocols like RMI (Remote Method Invocation) to collaboration development toolkits like Habanero®. One of the more recent and intriguing of these is JavaSpaces™.

JavaSpaces is a new high-level API for sharing objects over a network developed by Sun Microsystems, creators of Java. Its simple, yet powerful approach to network communication has drawn a significant amount of interest, but it is still a fledgling technology. Currently, a team at the Information Directorate of the Air Force Research Laboratory (AFRL) at Rome, New York is researching the possibility of integrating a large quantity of military information resources into a JavaSpaces-based network, under the ambitious Joint Battlespace Infosphere (JBI) project. The JBI is a Department of Defense initiative to develop a system to make military intelligence data from various sources more easily accessible to commanders [10]. However, tools developed for the JavaSpaces API are still scarce, to say the least, so it would be of interest to develop a proof-of-concept to show that it is practical to utilize JavaSpaces in such a project.

Clearly, collaboration would be an area of great interest when working with a large collection of military information, to be able to better understand and use the information. In addition, the sheer quantity of data requires the application of information visualization techniques so that it may be readily understood and acted upon. This is especially important when dealing with megabytes of raw simulation output data, which is all but meaningless until processed into some visual form.

An important property of a synchronous collaborative tool (that is, one in which the participants are working together at the same time) is interactivity. A collaborative tool will be awkward and unwieldy if response times are not nearly immediate; interactive response times for simple, frequent tasks should be in the range of approximately one second or less [13]. Thus, a primary concern when developing this application will be the performance of the JavaSpaces API. Since the API is at such a high level, it is expected that the performance might suffer as a result; this is an issue of concern for this research.

1.1.2 Information Visualization

Though information visualization has been around since the first time someone drew a picture to better understand a situation, it has lately been closely associated with computers for several reasons. Not only have computers become ubiquitous in modern life, but now powerful processors capable of rendering detailed three-dimensional images in real-time are available at affordable prices to just about anyone, thanks to the recent revolution in graphics cards.

In addition, computers add a new dimension to visualizing information that was previously impractical: users can now interact with the visualization. This opens a whole new world beyond the traditional pie charts and bar graphs. Users can now specify viewpoints in 3-D scenes, quickly filter the information being displayed to the desired level of detail, modify the visualization's parameters to their own tastes, and much more.

Information visualization is so deeply ingrained into computer systems that we take much of it for granted. For instance, graphical operating systems are a way to visualize the complicated functionality of a computer without overwhelming the user. WYSIWYG (What You See Is What You Get) word processors can be considered to utilize information visualization.

When dealing with complex sets of data that influence time-critical decision-making, it is important to be able to convey the information to the user in an efficient and accurate manner. A great deal of work has been done researching elegant and flexible techniques for displaying assorted classes of data, and several groups have developed toolkits implementing these techniques. Stuart Card's *Readings in Information Visualization* provides a compilation of such research [3]. It would be useful to design or adapt such a toolkit for any visualization framework, in order to further speed the application development process.

1.2 Application Frameworks

A technique that has been used for over twenty years to facilitate the development of large software applications is the application framework. A framework is commonly defined as "the skeleton of an application that can be customized by an application developer." [6] It defines a set of interfaces, abstract classes, and possibly even fundamental concrete classes representing the common elements of a given type of application. These components can then be reused between different applications of the same category in order to save time and effort.

1.3 Problem Background

Rome Laboratory produces a variety of simulations, such as weather data and chemical plume analyses. The output of these simulations tends to be in the form of a nigh-unintelligible gridded data set, possibly in binary form. Of course, if one is to make any use of this data, it will have to be visualized in some way. These data sets can be large enough to be too unwieldy to transmit across networks regularly, particularly across a relatively slow Internet connection. Thus it is impractical to share the actual data set with remote users who wish to view it.

Clearly, the data set must be processed or filtered in some way on the server side to reduce network traffic. This could mean constructing the visualization on the server and sending it to the clients, or reducing the data set to just that which is relevant to what a user wishes to examine. This problem will have to be addressed when developing a collaborative visualization tool designed to operate over an arbitrary network.

In order to demonstrate the functionality of the framework developed in this research, an application will be created to visualize the output of a chemical plume simulation and allow collaborative interaction with the visualization. In addition, the issue of network bandwidth restrictions will be addressed.

1.4 Problem Statement

The goal of this research is to develop a highly modular, flexible collaborative visualization framework and toolkit utilizing the JavaSpaces API. The product should be, at its core, generic enough to handle a broad class of visualizations and collaboration methods without reducing to nothing more than a set of interfaces lacking any substantial supporting code. Ideally, an application developed within this framework should achieve interactive response times, if the JavaSpaces API is capable. To demonstrate the functionality of the framework, an application will be developed within it, operating on data produced by chemical plume simulation software provided by Rome Laboratory. This application may serve as a proof-of-concept model for the migration of a portion of the Joint Battlespace Infosphere to JavaSpaces.

1.5 Research Objectives

The ultimate goal of this research is to produce an aid to the development of JavaSpaces applications for visualizing various data sets within a collaborative environment, without inordinate sacrifice of flexibility, performance, or robustness. Should this prove unattainable, a secondary objective would be to provide feedback on the appropriateness

of JavaSpaces as a backbone for communicating large numbers of complex data sets in interactive environments, as well as providing a stepping-stone for future efforts in this area of research.

1.5.1 Framework Design Goals

The idea to develop a collaborative visualization framework was inspired by Kurkowski's thesis detailing a general information visualization framework [11]. Thus, the goals of this research would be similar to those found in his thesis, along with addressing any perceived shortcomings.

In his thesis, Kurkowski enumerates the following goals:

- Support decomposition of functionality
- Facilitate reuse in future software development
- Develop visualization components that improve upon existing analysis capabilities

In addition to the above goals, this research aims to utilize and demonstrate the viability of the JavaSpaces technology, as well as address the issue of network bandwidth consumption when transmitting large data objects.

The primary shortcoming noted in the design of Kurkowski's visualization framework was a lack of flexibility. Any application built using the framework would have the same set of user interaction widgets, though the developer could customize the interface to some degree by adding drop-down menu items. In addition, certain interaction

functionality was pre-defined, notably setting a left-mouse-click to re-center the viewing window, and automatically including zooming functionality, whether it was appropriate for the application or not.

A possible reason for the inflexibility of the framework is that it aimed for too broad a class of applications (visualizations). As a result, extraneous code was forced into the framework where it did not belong in order to lend it substance. In an attempt to correct this shortcoming, this research is targeted at a narrower subset of applications: collaborative visualizations utilizing JavaSpaces. A reasonable degree of flexibility is a major consideration in the design of the framework.

In addition to framework design goals, there are certain considerations associated with visualization and collaboration that must be taken into account. Visualizations must be able to respond to user input promptly to maintain a sense of interactivity. Any collaborative application must provide some means of verbal/linguistic communication, since it is the natural way for humans to communicate in society. To rob users of that ability would frustrate them and hinder communication and, consequently, collaboration.

1.5.2 Scope and Limitations

The resulting framework and toolkit will be aimed at visualizing simulation output data from Rome Labs; specifically, a tool for visualizing chemical plume data will be implemented using this framework. The data sets to be used will be supplied by Rome Laboratory, and are expected to be actual simulation output. The toolkit will not be

particularly extensive, but should include basic collaboration utilities such as a chat room, whiteboard, and logging capabilities, as well as some visualization tools. Voice and video communication facilities would likely be best achieved through COTS software or other means, rather than a JavaSpaces-based tool, since JavaSpaces appears ill suited to streaming data. Network communication with JavaSpaces is accomplished through sharing Java objects, which would be awkward to adapt for streaming media.

1.6 Document Overview

Within chapter two is in-depth discussion of related background material, including related work in the field of collaborative/visualization frameworks. Chapter three covers the design and implementation of the framework itself, with some discussion of its utility, including a description of the interface. Chapter four discusses the application implemented using the framework, noting the benefits gleaned from using the framework over coding the entire application from the ground up. The results and analysis of some basic performance testing is also found therein. Finally, chapter five contains conclusions and a brief summary of the research.

Chapter 2 – Background

2.1 Collaborative Computing

According to Michael Schrage in his book, <u>Shared Minds</u>, collaboration is a purposive relationship, at the heart of which rests a desire or need to solve a problem, create, or discover something, within a set of constraints. The constraints he brings up are expertise, time, money, competition, and conventional wisdom (upon which he elaborates, "the prejudices of the day"). [12]

While most of us envision collaboration as something that takes place among several people working together, conversing, taking notes, and building on each other's ideas, Jason Wood, in his PhD thesis on collaborative visualization, discusses the easily-overlooked aspect of asynchronous collaboration, which occurs when several people interact at different times, for instance, leaving messages on a bulletin board or sending emails. [14]

With collaboration more clearly defined, what tools make for more effective collaboration? Schrage goes on to point out that a shared space to work with is vital. A blackboard, for instance, has been an effective collaboration tool for centuries, with little modification, but it has its obvious limitations. For instance, it has limited space, and there is no record kept of passages that have been erased to clear room. Furthermore, words written on a blackboard can often be nearly or completely illegible, and rearranging items is inconvenient. The conversation that accompanies a session of

collaboratively utilizing a blackboard is ephemeral, and is lost except for the fallible memory of the participants. [12]

A computer-based collaborative tool can solve several of these problems. The limitation of space can be solved in various ways, from zooming in and out to simply opening more windows as needed. Computers can maintain records of past activity in persistent memory, and written annotations can be typed instead of hand written to eliminate the problem of legibility. In addition, computers can automate the rapid generation of complicated visual representations of data, which otherwise could not be done in a real-time setting. Thus, there has understandably been a great deal of interest in utilizing computers as tools to improve collaboration. [12]

2.2 Visualization Techniques

A significant amount of research has been invested in developing generalized techniques for visualizing a wide range of data sets in order to avoid having to design a custom visualization for each application. The package developed in this research includes a small toolkit implementing some of these various techniques.

2.2.1 Parallel Coordinates

Alfred Inselberg describes a method of visualizing data sets consisting of more than three dimensions on a two-dimensional display medium in his paper "Multidimensional Detective". He describes a simple technique that involves plotting data elements as lines on a graph, whose axes run parallel to each other. These "parallel coordinates" offer

several benefits. Amongst those, Inselberg cites a low representational complexity, the flexibility of being able to represent any number of dimensions, and the uniform treatment of each variable. [8]
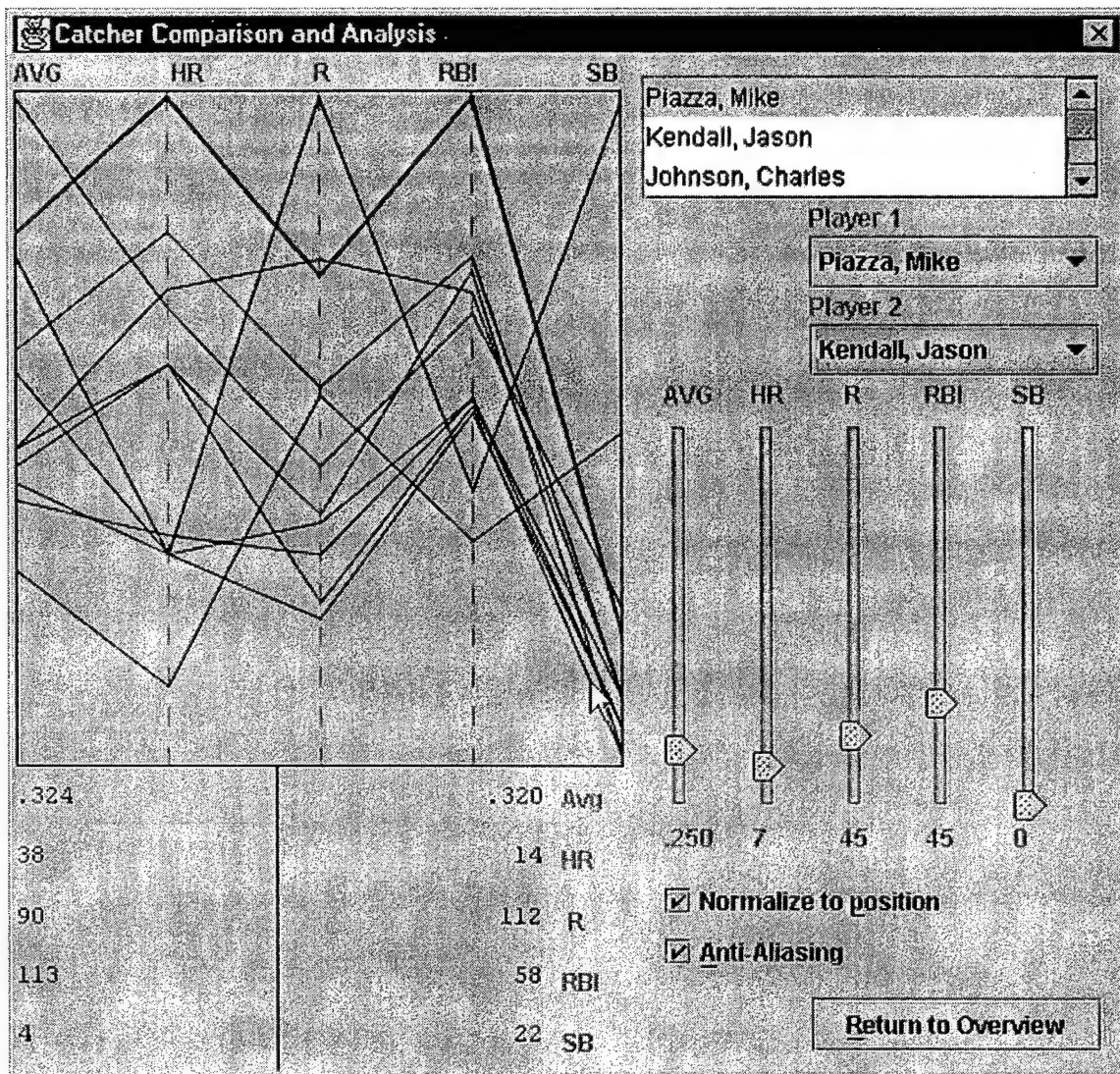
Figure 2.1 An application utilizing parallel coordinates to represent five-dimensional data about major league baseball players' performance in the 2000 baseball season.

An example of an implementation of the parallel coordinates data visualization technique is shown in Figure 2.1. In this example, five commonly referenced baseball statistics are represented for a set of professional baseball players. It is an effective representation for discerning overall trends and patterns in the data set, such as the observation that

catchers, with few exceptions, generally tend to have very few stolen bases (SB), and that one catcher stands out from the pack in the categories of home runs (HR) and runs batted in (RBI). Inselberg uses a 16-coordinate data set to demonstrate the technique in his paper, illustrating the flexibility of the method [8].

2.2.2 Other Basic Techniques

Other less exotic visualization techniques are also found in the toolkit, including simple line and bar graphs. While not the product of cutting-edge research into visualization, they have long been a standby for visualizing uncomplicated data sets.

2.3 Application Frameworks

An application framework illustrates the architecture of a software system in order to make development easier. It describes the objects and interactions within the system, saving time in the analysis and design of an application. In essence, this effectively reuses the high-level design work of software development. In addition, a framework will often provide a significant amount of functional code, which also speeds the implementation phase. [6]

Further projects can reuse components from earlier systems, since they were developed using the framework's defined interface. This not only saves time programming, but can also enhance the quality, performance, reliability, and interoperability of software. This is due to the fact that once a module is finalized, it is likely to have undergone a great deal of debugging and refinement, and thus can generally be considered stable. This in

13

turn allows programmers to concentrate on new modules when attempting to isolate problems with an application in development. [6]

Frameworks take advantage of all of the primary advantages of object-oriented programming [6]:

Data abstraction– Frameworks contain objects of abstract data types whose implementations may vary behind a common interface.

Polymorphism– A framework allows a developer to mix and match different components and makes it possible to develop generic components to work with varied objects.

Inheritance– Frameworks typically provide abstract base classes from which custom components are derived, which makes development of new components easier.

2.4 The Jini and JavaSpaces Technologies

The Jini Connection Technology is envisioned to facilitate "impromptu networks" of heterogeneous devices [9]. In theory, once a Jini network is set up, a user should be able to plug in a Jini-enabled network device, such as a PDA (Personal Digital Assistant), wireless phone, or printer, and expect it to immediately be able to access the network and the assorted services available on it, without requiring any time-consuming set-up and configuration.

In addition, the Jini network should robustly handle the dynamic environment that is the product of network devices constantly entering and leaving the community or otherwise changing in state. Aside from changing network topology, Jini also addresses the problems of network reliability, latency, bandwidth limitations, and security. While Jini is written in Java, any programming language can theoretically interact with a Jini network, as long as there is a piece of Java code that describes the interface to the service provided. [5]

JavaSpaces is a simple, high-level network API for Java, built upon Jini. It offers a remarkably elegant interface for network programming, with only four major methods for interacting with the shared space: read, write, take, and notify. JavaSpaces abstracts network communication into a virtual shared memory space, referred to as a JavaSpace. Applications communicate indirectly through this space; a "sender" writes an object into the space, while any interested party can then read the object from the space. [7]

The *read* and *take* methods both allow an application to access an object in a JavaSpace, but the *take* method removes the object from the space. This handles concurrency issues, as once an application takes an object from a JavaSpace, it can be sure that no other processes will modify the object while it is being used. Once a process that has taken an object from a JavaSpace is finished with the object, it can then write it back to the JavaSpace. The *notify* method sends an event to a process when an object matching a given "template" enters the space. [7]

An application can find objects that it is interested in by creating a template. This template is an instance of the object it wishes to read from the JavaSpace. It sets the attribute fields of the template to values matching the characteristics of the object it is looking for, using a null value as a wildcard. For instance, an application could look for a Faculty object whose department attribute is set to "Computer Science" by setting the corresponding attribute in the template, leaving all other fields null. The template is then passed into the *read*, *notify*, or *take* method, as appropriate. [7]

2.5 Existing Frameworks

There are several existing software frameworks for collaborative computing or information visualization, but no known frameworks that deal with both areas.

2.5.1 Kurkowski's Information Visualization Framework

In 1999-2000, Stuart Kurkowsi developed a framework for information visualization applications, which was the subject of his thesis [11]. His framework was written in C++ and utilized the Fox API for GUI implementation. While his thesis investigated an interesting area of research, the finished product came up short of his goal, in that it was appropriate only for a very limited set of visualization applications that had nearly identical user interfaces.

A possible reason for this result is that the class of applications that he selected was too broad to be able to factor out a great deal of common code, but rather than leave most of the code to the specific application, he moved more of it to the framework level than was

reasonably justifiable. He moved the majority of the GUI components to this level, allowing user control of only pull-down menu items. This could often result in leaving useless GUI components in the application or artificially restricting an application designer to a specific set of GUI controls.

2.5.2 NCSA Habanero

Habanero is a framework for collaboration applications written in Java [4]. It functions by providing state and event synchronization for multiple copies of a Java software tool. Habanero replicates an application across all clients using it, then maintains a consistent shared state though the sharing of state-changing events, which are defined by the application programmer. Habanero ensures that all such events are seen by all clients in the same order. In addition, it provides an arbitrator mechanism that restricts what actions can be performed by which clients at any given time. Unfortunately, this arbitrator makes socket connections to each client, which does not scale well. Because of that, a multicast implementation is being investigated. Habanero provides a set of tools including text and voice chat, a drawing tool, and a shared whiteboard, as well as record and replay functionality.

The primary disadvantage to Habanero is that it displays the same screen to all participants. This means that all users of the same application will have the same interface and see the same information. This can be a significant limitation when different users of the application have different roles.

2.5.3 Java Shared Data Toolkit (JSDT)

JSDT is Sun Microsystems' collaborative application framework offering [2]. It handles message passing through a session abstraction and several shared data structures/mechanisms, including shared byte arrays, token-based distributed synchronization, and object-passing through serialization across a channel. JSDT provides the functionality to manage objects, which allows the application to control participant access to certain objects. A JSDT application can be implemented using any of a number of underlying communications protocols, either TCP sockets, RMI (Remote Method Invocation), HTTP, or LRMP (Lightweight Reliable Multicast Protocol).

2.5.4 Tango Beans

Tango Beans, developed by Lucasz Beca, Geoffrey C. Fox, and Marek Podgorny at the Northeast Parallel Architectures Center, is also a collaborative application framework [1], but unlike JSDT and Habanero, it provides a component-based API, based on JavaBeans™. Tango Beans addresses the limitation of simply sharing display of the application, which is what Habanero provides. The authors of the Tango Beans paper illustrate this advantage by giving the example of an application in which a student and teacher might have different interfaces. Since Tango Beans is based on JavaBeans, it is deployed on the web as an applet, and uses the same mechanism as JavaBeans to construct an applet from individual beans.

18

2.6 The Rome Laboratory Joint Battlespace Infosphere Effort

The Information Directorate of Air Force Research Laboratory at Rome, NY has several groups working on a Department of Defense project known as the Joint Battlespace Infosphere, or JBI. This project is envisioned to make militarily interesting data available from various sources to any commander who needs access to it. One source of data is simulation output of, for example, chemical plume behavior forecasts.

One team at Rome Laboratory is working on the problem of representing this data in a format convenient for efficient transmission across a network and utilization in assorted client applications. They are attempting to port their application code to the Java programming language in order to have a common object format, and are interested in investigating the possibility of using JavaSpaces as the underlying network protocol, since Jini is already being configured and installed on the network, and JavaSpaces seems well suited to their application.

JavaSpaces is a fledgling technology, and is still relatively unproven in industry. Thus, it would be of interest to this team to develop a proof-of-concept application to examine the aspects of the JavaSpaces API as well as explore other issues dealing with handling simulation output data in network applications.

2.7 Summary

The goal of this research is, in short, to create a JavaSpaces-based framework for collaborative visualization applications. To demonstrate its functionality, a proof-of-

concept application for Rome Laboratory will be implemented. The purpose of the application is to investigate the viability of utilizing JavaSpaces to share simulation output data across a network, as well as to test the research framework. In this chapter, background information regarding general, visualization, and collaborative application frameworks has been presented, along with material explaining the Jini and JavaSpaces technology, as well as the motivation for the proof-of-concept application.

Chapter 3 – Framework Approach and Design

This chapter discusses the design and implementation of the underlying JavaSpaces-based collaborative visualization framework. Topics covered include framework design considerations and a detailed implementation overview and interface definition.

3.1 Framework Design Overview

Analysis of Kurkowski's visualization framework revealed that there was little framework functionality that could be implemented for visualizations without sacrificing a great deal of flexibility. On the other hand, a significant amount of work could be done in the interface between a visualization and the collaborative/network component. Thus, the vast majority of the framework elements described here deal with this aspect. Visualization development aids are better restricted to a toolkit.

The core of the framework is implemented as the ColVis (**Col**laborative **Vis**ualization) class. This class handles all network interaction, and is the application developer's primary interface to the framework functionality. It maintains a shared communication channel in an existing JavaSpace, as well as a set of visualization modules specified by the application developer. The visualization modules can send messages to the space and be notified of new messages via the ColVis class. The flow of network messages is illustrated in Figure 3.1. Limited additional functionality is supported, primarily state-awareness functions such as the ability to retrieve a list of current visualizations or participants, and user state functions.
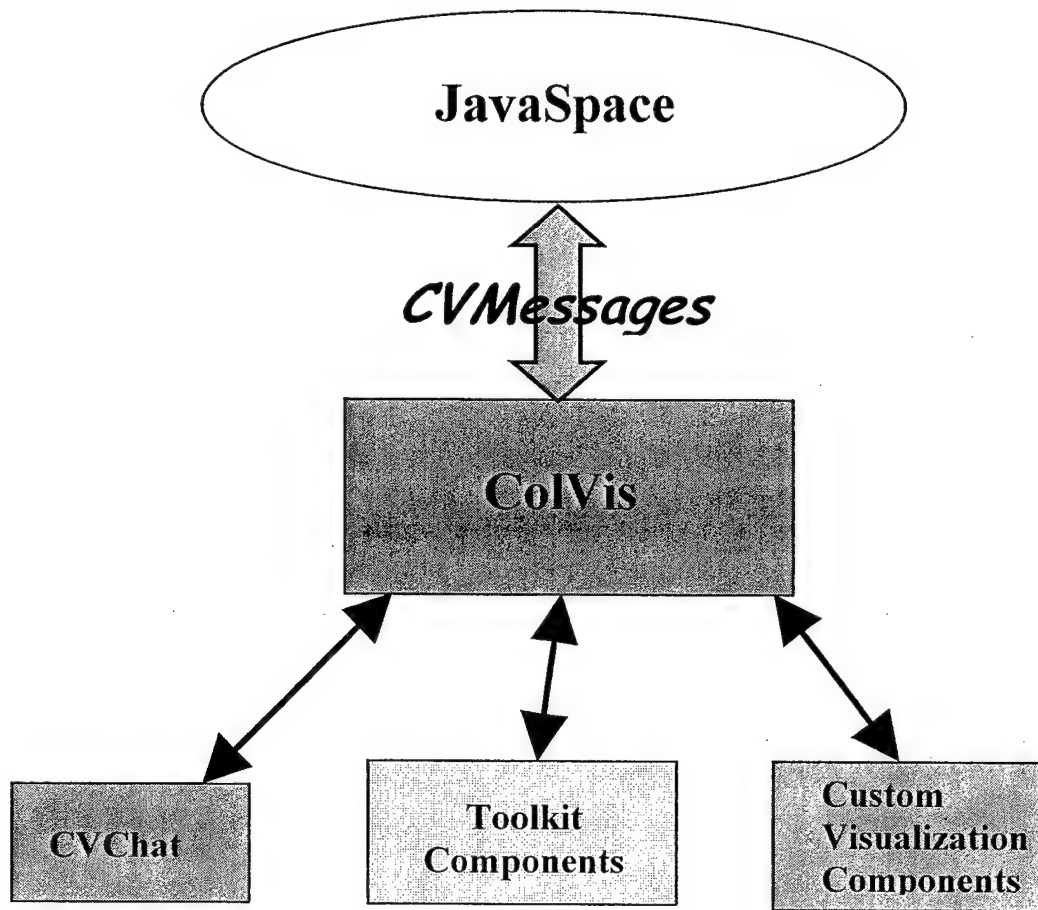
Figure 3.1: Conceptual Framework Overview

3.2 Framework Implementation Detail

There are three classes in the framework that are important to understand in order to implement visualization applications. The ColVis class is the backbone for network communications and maintaining visualization-independent state. The CVMessage class is used for representing messages passed across the network via the JavaSpace. The CVVisualization class is the base class from which all visualization modules inherit. In order to help reduce the network bandwidth consumption of large data objects, a

DataWrapper class has been included which allows the user to easily compress the contents of a raw data file. Each of these classes are described in detail, accompanied by a table defining their key public methods, in the following sections. A partial class diagram of the ColVis framework is shown in Figure 3.2.
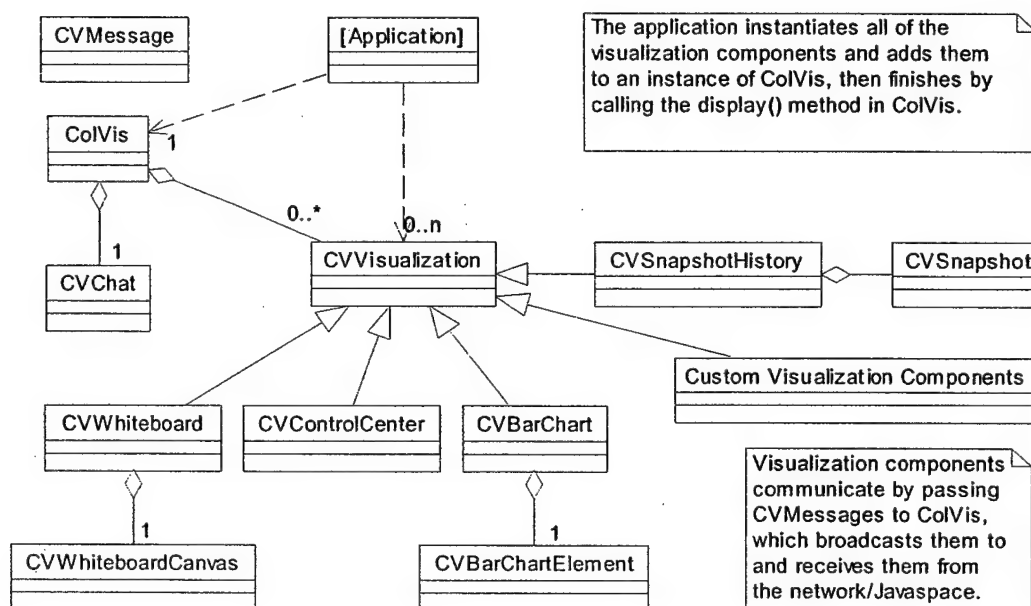

Figure 3.2: Partial Framework Class Diagram

3.2.1 The ColVis Class

The ColVis class sends messages to and receives messages from the shared JavaSpace. In addition, it handles any moderator functionality, maintains a list of session participants and active visualization modules, and displays the chat and visualization frames.

23

The ColVis class can be used to develop collaborative visualization applications using very few methods. At the most basic level, one can simply use the default constructor to instantiate an unmoderated ColVis, then add visualization modules with calls to *addVisualization(CVVisualization).* In this way, the developer adds each visualization module instance (custom or from the toolkit) that is to be included in the application, and then finally calls *display()* to create the graphical user interface (GUI). Other methods are provided for more advanced functionality (see Table 3.1), but one could use only the few aforementioned methods to create a very capable application.

Table 3.1: ColVis Public Methods of Interest

ColVis(boolean isModerated) ColVis()	Constructor takes boolean parameter defining whether the application is moderated. Defaults to unmoderated.
void addVisualization(CVVisualization) void removeVisualization(CVVisualization)	Add a visualization instance to be displayed as a tab in the visualization frame (CVVisframe), or remove an existing visualization. display() must be called to effect changes in the GUI.
void display()	Display application GUI, once visualization modules have been set.
boolean isConnected()	Returns true if application is currently connected to a channel.
boolean isModerator() boolean isModerator(String participant) boolean canSpeak() boolean canSpeak(String participant)	User state accessors used when a channel is moderated. A moderator can change the state of another participant, and a muted participant cannot contribute to a collaborative session by sending any messages to the JavaSpace. The default methods return the state of the local user.
Vector getVisualizations() CVChat getChat() void setChat(CVChat) Vector getParticipantNames() int getNumParticipants() String getName() void setName(String)	Assorted available accessors and mutators The getVisualizations() method returns a Vector of CVVisualizations, and the getParticipantNames() method returns a Vector of Strings.

24

3.2.2 The CVMessage Class

The CVMessage class forms the basic message objects that are passed across the network via the JavaSpace. Due to restrictions of the JavaSpaces technology, all its fields are public, so they can be accessed and changed directly. Of particular note are the *sender*, *type*, and *content* fields, all of which are of type java.util.String. The *sender* field is usually automatically set by ColVis to the name of the local user whenever a visualization module sends a message. The *type* and *content* fields are usually set by the constructor when the CVMessage is instantiated. These are used as a quick and convenient means to determine the purpose of a message. The simplest messages can be defined using only those two fields, or, in some cases, just the *type* field. However, a mechanism exists for creating more elaborate messages. A CVMessage may have any number of java.lang.Object parameters, which are set using the *addParameter(Object)* method, and later accessed using the *getParameter(int index)* method. The key methods of CVMessage are summarized in Table 3.2.

Table 3.2: CVMessage Public Methods of Interest

Method	Description
CVMessage(String type, String content)	Constructor
void addParameter(Object) void removeParameter(Object) Object getParameter(int index)	Methods to add/remove java.lang.Object parameters to the CVMessage, which can be accessed later, by index, using the *getParameter* method.

3.2.3 The CVVisualization Class

The CVVisualization class is the abstract base class, extending the JPanel class from the Java Swing package, from which all visualization modules will inherit. It contains methods which simplify the communication with the network for the user. Encapsulated behind an interface method, it uses an instance of ColVis (assigned in the constructor or via a mutator method) to send and receive network messages. The class also maintains basic user interface attributes of the visualization, such as the title as it appears on its tab, as well as whether or not the visualization is shared (that is, whether or not it is allowed to send and receive messages via the JavaSpace).

Since CVVisualization is a child of the JPanel class, its children can also use the public methods found therein. The appearance of the visualization module is determined by laying out graphical components within it just as one would with a JPanel in Java's Swing package. The key public methods of CVVisualization are summarized in Table 3.3.

Table 3.3: CVVisualization Public Methods of Interest

Method	Description
CVVisualization() CVVisualization(ColVis) CVVisualization(ColVis, String title)	Constructors. A CVVisualization will default to being shared unless the no-argument constructor is used. An assigned instance of ColVis is necessary to send and receive network messages.
void sendMessage(String type, String content) void sendMessage(CVMessage)	Sends a message to the channel across the network, via the assigned instance of ColVis, if this CVVisualization is shared.
String getType()	Returns the String "Visualization" if not overridden. Can be used in conjunction with the *getVisualizations()* method in ColVis to form a list of instances of a particular type of visualization module, by iterating through the complete list and checking the return value of *getType()*.
abstract void notify(CVMessage)	Must be implemented in child class. Defines how the visualization module handles incoming CVMessages from the network. Whenever a new message arrives, ColVis will call each CVVisualization's *notify(CVMessage)* method, passing in the incoming message. Can be implemented as an empty function, which indicates the module ignores all incoming messages.
boolean isShared() void setShared(boolean) String getTitle() void setTitle(String) Icon getIcon() void setIcon(Icon) String getToolTipText() void setToolTipText(String)	Assorted available accessors and mutators The title, icon, and tool tip text affect the appearance of the visualization module's tab in the application. As noted earlier, a module that is not shared cannot send or receive messages via the network.

3.2.4 The DataWrapper Class

The DataWrapper class was created to address the issue of network bandwidth concerns when transmitting large data objects. It allows the user to easily convert a raw data file into a compressed format, which is serializable, so that it may be written into a JavaSpace. To utilize the DataWrapper class, one implements a derived class that provides a method to return a Java object created from parsing the raw data, found in a

27

protected byte array field called *data [ ]*. A summary of the key methods in the DataWrapper class can be found in Table 3.4.

Table 3.4: DataWrapper Public Methods of Interest

Method	Description
DataWrapper() DataWrapper(String dataFileName)	Constructors. If you pass a filename as a parameter, it will automatically load the specified file.
void readFromFile(String dataFileName)	Load specified file
void zip()	Compress data
void unzip()	Decompress data

3.3 The ColVis Toolkit

The toolkit classes were created to extend the basic capabilities provided in the core framework, as well as providing added code reuse benefits. Some module types will be commonly found in many collaborative and/or visualization applications. Some of these general-purpose components were implemented in a toolkit to be packaged with the framework.

Perhaps the most important 'utility module' is the CVWhiteboard. In addition to providing the classic collaborative functionality of a whiteboard, it also serves as a means to easily display and annotate visualization output. Among its features are text annotation, icon/stamp creation, and the ability to load background images.

The CVSnapshotHistory is a complementary component to the CVWhiteboard. It allows the user to capture "snapshots" of the whiteboard at any time, which can later be restored to any whiteboard. This allows the user to copy an image from one whiteboard to

28

another, create a running history of the whiteboard session, or restore a whiteboard to an earlier state if unwanted changes have been made.

Other toolkit components are included, including basic visualization modules and a control center to dynamically add or remove whiteboards and snapshot history modules at runtime.

3.4 Implementing a ColVis-based Collaborative Visualization Application

In order to implement an application using this framework, first one creates any number of visualization modules by inheriting from the base CVVisualization class. Then, the user instantiates them, adds them to an instance of ColVis, and finally calls the *display()* method in ColVis. This process is generally best placed in the *main* method of a class consisting of nothing else. An example of the code required to set up the application using a pair of modules found in the toolkit is found in figure 3.3.

```
public class CVApplication
{
    public static void main(String [] args)
    {
        ColVis cv = new ColVis();

        CVWhiteboard wb = new CVWhiteboard(cv);
        wb.setTitle("Main Whiteboard");
        cv.addVisualization(wb);

        CVSnapshotHistory history = new CVSnapshotHistory(cv, wb);
        history.setTitle("Snapshots");
        history.setShared(false);
        cv.addVisualization(history);

        cv.display();
    }
}
```

Figure 3.3: Code for a simple ColVis-based application

29

As seen in figure 3.3, it requires fewer than a dozen lines of code to implement this simple, yet non-trivial ColVis-based application. The above application creates a whiteboard and a snapshot history, which can capture "snapshots" of the whiteboard at various points in time and restore them later, if the user wishes. Any changes to the whiteboard are propagated to the whiteboards of all other users on the same channel using this application. In addition, it was deemed necessary that some means of verbal communication be included in any collaborative application. A simple chat module was chosen to integrate into the framework over other methods of verbal communication for several reasons. As mentioned in the first chapter, JavaSpaces was not designed for streaming media. In addition, text chat uses very little bandwidth, is intuitive to use, and is easy to implement.

3.5 Summary

The framework allows visualization modules to be developed independently of each other and assembled in various combinations to form different collaborative visualization applications. Ideally, the functionality of the modules will complement each other, resulting in an application whose overall power is greater than the sum of its components. This high degree of modularity addresses the goal of functional decomposition.

Another goal taken into consideration in the design of the framework was the principle of software reuse. The very nature of the framework facilitates code reuse, as well as design reuse. Every application developed using this framework will reuse the network communication code found in the ColVis class, as well as the chat module. In addition, it

is likely that toolkit modules or possibly even custom components could be reused in different applications, as well.

The DataWrapper class was developed in response to the research goal of addressing the issue of limited network resources. This class helps solve the excessive bandwidth problem associated with sending large data objects across a network.

Maintaining a high degree of flexibility was a primary concern during the design of the ColVis framework. This was addressed by writing a minimal amount of visualization code into the framework, which primarily concerns itself with encapsulating the network communication. Thus, a visualization developer has little restriction on the appearance or function of a visualization module, besides those imposed by the Java Swing API. All visualization modules will appear within a tabbed pane within a single frame, and the network modules always appear the same, but this seems to be a reasonable tradeoff to significantly improve the simplicity and modularity of development.

Chapter 4 – Application Implementation

An application for collaboratively visualizing chemical plume simulation output data was developed in order to demonstrate the functionality of the framework. (See Figure 4.1) In addition to validating the functionality of the ColVis framework, this application was also intended to act as a proof-of-concept demonstrating the viability of JavaSpaces as a foundation for future network applications at AFRL in Rome, NY. This chapter describes this application and analyzes the performance of both the application and the framework.
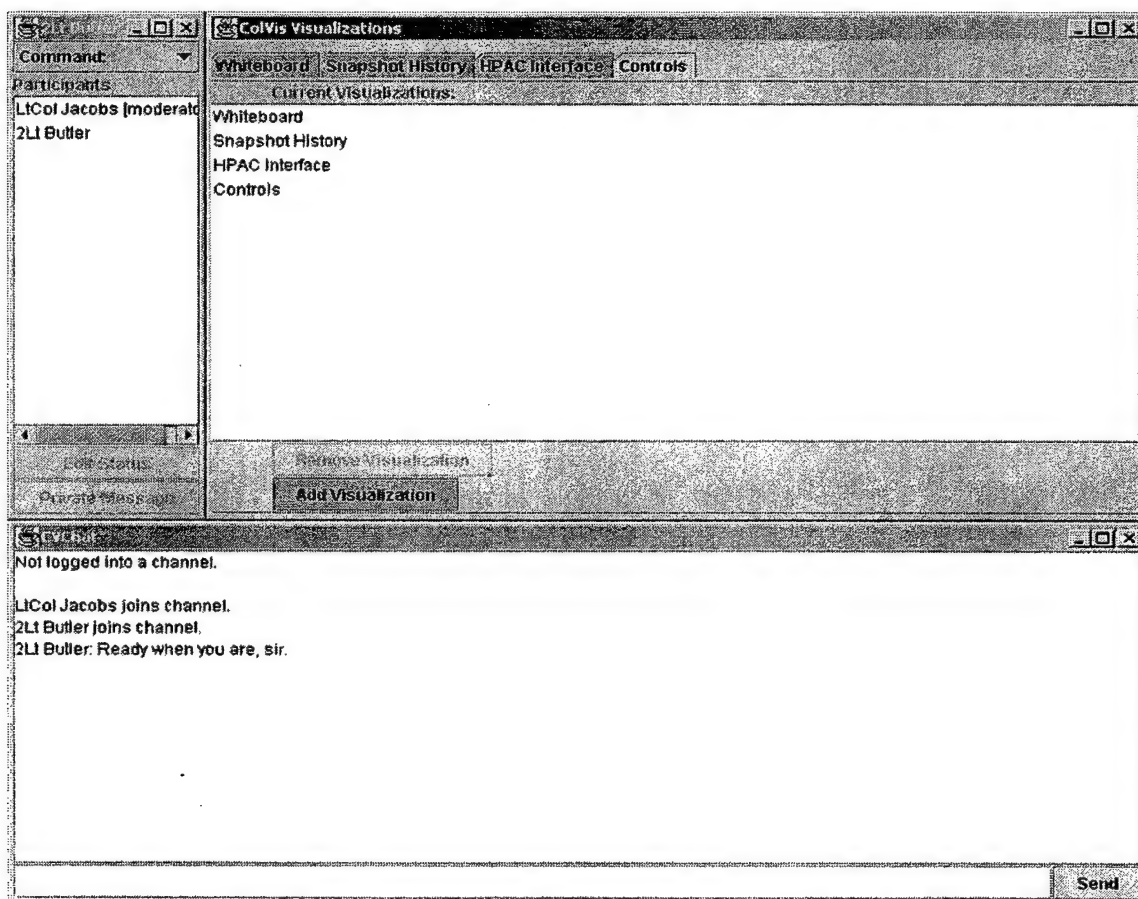
Figure 4.1 Demonstration Application- This 'Controls' module allows certain modules to be added and removed at run-time

4.1 Application Overview

The chemical plume collaborative visualization application consists of a number of general-purpose collaborative toolkit modules (a whiteboard, snapshot history, and visualization control panel) in addition to a specially designed visualization module. This custom module provides a user interface to the HPAC (Hazard Prediction and Assessment Capability) simulation software and generates a visualization of the resulting output. This module presents users with a simplified front-end (see figure 4.2) to HPAC, allowing them the ability to specify significant parameters for the simulation. If the module is designated as shared, then any participant may modify the simulation input parameters. When the simulation is executed, the module then takes the output data and generates a visualization of it, rendering it onto a whiteboard module of the user's choice (see figure 4.3). If the whiteboard is shared, all users who have a copy of the whiteboard will see the resulting visualization; otherwise it will only be displayed locally on the user's machine. Once a visualization is output to a whiteboard, it may then be subsequently annotated by any user that has access to it, using any tool available on the whiteboard, including text annotation, placing icons, drawing tools, etc.
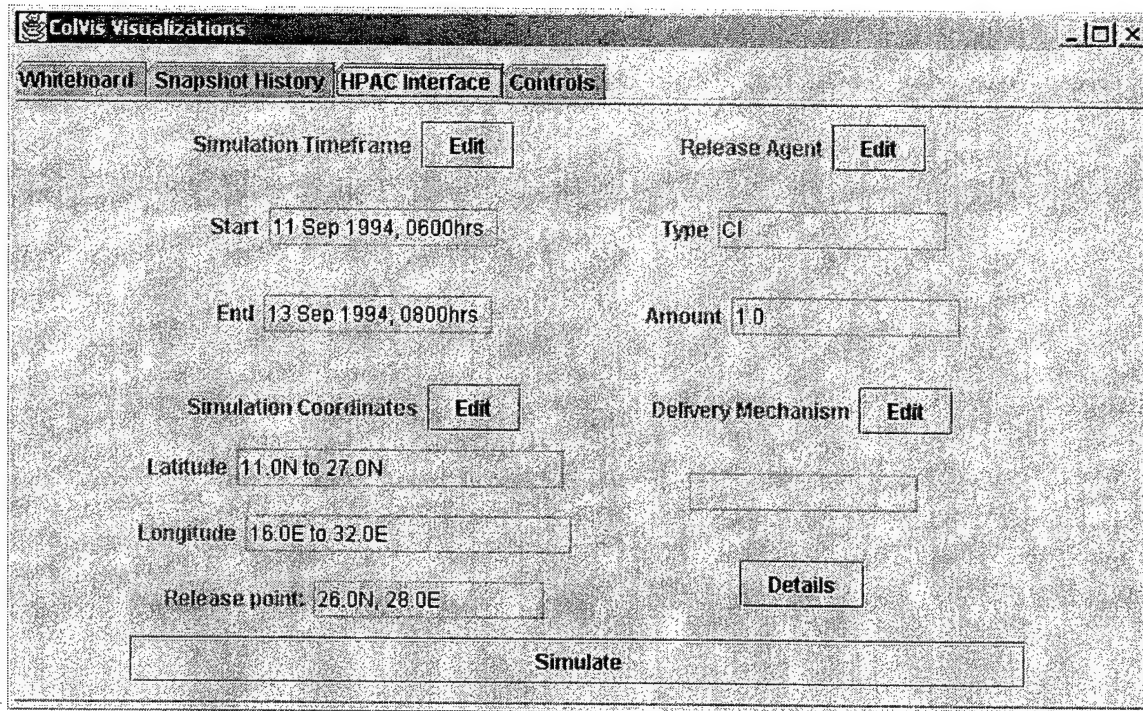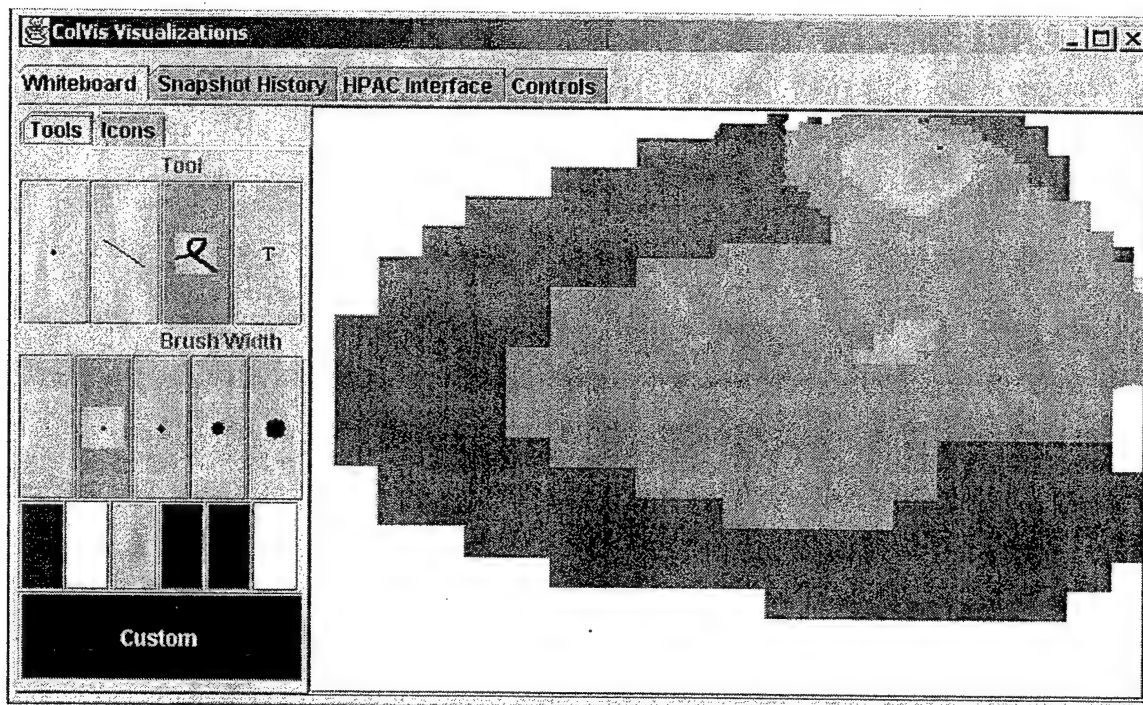
Figure 4.2 HPAC Interface Module

Figure 4.3 Plume visualization rendered on whiteboard

34

By utilizing the toolkit modules in conjunction with the chemical plume visualization, one can benefit from added functionality. For instance, one can create a thumbnail record of the simulation output over time. This is accomplished by adding one or more additional whiteboards using the control panel, creating a series of visualizations at various simulation times, and saving each visualization to the snapshot history. Later, any given snapshot can be restored to a whiteboard for further examination. A simpler utilization of the supporting modules would be to simply create a local whiteboard to make a set of annotations to, which can later be copied as a complete image to a shared whiteboard via the snapshot history. By doing this, the annotations are not viewed by everyone as they are made– the participants see only a finished, fully annotated image.

4.2 Framework Benefits

Given the framework and toolkit, the custom visualization module was significantly easier to implement. The network aspects were almost totally hidden, and the toolkit provided a great deal of supporting functionality, such as handling the display and collaborative modification of the resulting visualization image.

Reuse of the framework and toolkit components facilitated application development by reducing the amount of code that needed to be written; however, the benefit gained by the framework cannot be solely equated to the number of lines of code saved. In order to implement a custom visualization module, a programmer would need to first acquire an understanding of the ColVis framework and toolkit. Due to the simplicity of the

35

framework's design and interface, this would have been relatively straightforward, and certainly much easier than designing the functionality from scratch.

From the ColVis class itself, the programmer would need to know how to implement a basic application (see Figure 3.1 for an example), as well as understand the *getVisualizations()* method in order to obtain a list of whiteboards to select an output destination from. The programmer would also need to understand the format of the CVMessage class, which primarily consists of two String fields and a Vector of Object parameters—the constructor, *addParameter(Object)*, and *getParameter(int)* methods are all that need to be used. From CVVisualization, the programmer would need to use the *sendMessage(CVMessage)* method, as well as understand how to implement the *notify(CVMessage)* method, which are both fairly straightforward, as mentioned in the third chapter. In addition, the *getType()* method would have to be used to identify which visualizations are whiteboards. Finally, the programmer would need to use the *setImage(Image)* method in the CVWhiteboard class in order to output the visualization to a whiteboard.

Not including constructors, the programmer would need knowledge of approximately nine methods across four framework/toolkit classes in order to develop this application, plus any methods used to change the visualization module settings from the defaults (ie. *setTitle(String)*, *setShared(boolean)*, etc.). Most of these methods are exceptionally straightforward in usage and trivial to understand. Of course, the programmer needs to

36

be familiar with the Java Swing API, as well, in order to design the custom visualization module.

In addition to the time saved in the design phase of development, which is difficult to quantify, use of the framework resulted in over 50% code reuse for this application, by a conservative estimate. Since much of the code contained in the custom module consisted of simulator input parameters that were never used, but rather included for the sake of completeness and extensibility, that figure could easily be significantly higher.

4.3 Performance Analysis

Though there was some concern regarding the performance of JavaSpaces due to its high level of abstraction, rudimentary testing suggests that it is more than adequate. A series of tests were conducted to determine what level of network traffic a JavaSpace could sustain. For each test, one machine composed and sent a common type of CVMessage (a "draw line on whiteboard" message, which is composed of one Float, one Color, and four Integer parameters, in addition to the basic CVMessage characteristics) across the network as many times as it was able in five seconds, and reported the number of messages it was able to send and subsequently read from the network. Taking the lesser of the two values, a dozen tests were run on both a machine hosting the JavaSpace as well as a different machine on the same network. The results were somewhat surprising.

The tests on the local machine averaged 131.6 messages in the five-second span (26.3 messages/second), and consistently reported an equal number of received messages as

37

sent messages, indicating a very low latency. The tests on the remote machine, however, averaged 175.0 messages in the five-second span (35.0 messages/second), at times with a small discrepancy between sent and received messages. The remote machine, while suffering from slightly greater latency than the local machine, was actually able to compose and send more messages per second. This appears to indicate that the throughput limitation for this particular type of message, each of which amounts to 665 bytes of data, is not JavaSpaces itself, but rather the CPU time available to the computer, since the remote machine, identical to the local machine, was running fewer processes in the background, as it was not hosting the JavaSpace and its associated services. If this hypothesis is correct, if you increase the message size, there would be a point at which the CPU ceases to be the bottleneck, and the message rate would begin to drop. However, the message size used in this test, containing six extra Java Object parameters, is probably representative of a typical high-volume message. In this case, regardless of the bottleneck, 26-35 messages per second on a LAN should be more than adequate for most small-scale applications, though further testing is needed to determine scalability.

The other potential performance bottleneck occurs with very large messages, particularly image files. A typical such message (e.g. a whiteboard-sized image) would be of the magnitude of 600kB. However, these types of messages would likely be quite rare, and still do not introduce an unbearable delay on a high-speed network. This type of message is generated when an image is placed onto a whiteboard, as when a snapshot is restored or plume visualization is generated. One could possibly reduce the image (and consequently, message) size by encoding the image into a compressed format. However,

that would require significantly more complex code, and could create problems related to losses incurred in the compression process, depending on what algorithm is used.

If network bandwidth is at a premium, and there is processor time to spare, the plume visualization module could be modified to transmit the raw data file across the network instead of the final visualization. Each participant would then generate the visualization independently. While the raw data file can be 150kB or larger in size, it can be compressed using an implementation of the abstract DataWrapper class that is contained in the toolkit to help address network bandwidth issues, a concern when dealing with gridded scientific data such as the plume simulation output. As an example, a 173kB ASCII data file was compressed down to 16.8kB using an implementation of this class, a greater than 10:1 compression ratio. The overhead of a CVMessage is approximately 351 bytes (the size of a message without extra parameters), which is essentially negligible.

4.4 Summary

The demonstration application, consisting of several ColVis toolkit modules and a custom visualization module developed as a collaborative interface to third-party chemical plume simulation software, performed to optimistic expectations, and was considerably easier to implement using the ColVis framework than it would have been otherwise. Also, JavaSpaces does not appear to be a hindrance to network performance based on limited testing, though additional testing to determine scalability is needed. Finally, the DataWrapper class serves as a potential means to reduce network bandwidth

usage by acting as an abstract base for developing data objects containing significantly

compressed raw data.

Chapter 5 – Conclusion and Summary

The overarching goal of this research was to create a viable framework for the development of collaborative visualization applications utilizing JavaSpaces. Additional goals were to devise means of streamlining network traffic to conserve bandwidth and to demonstrate the viability of JavaSpaces as a networking API. This chapter summarizes how each of these goals was achieved.

5.1 The ColVis Framework

Several goals were kept in mind when designing the framework, as outlined in the third chapter. These were:

- Support decomposition of functionality

- Facilitate reuse in future software development

- Develop visualization components that improve upon existing analysis techniques of the sponsor (Rome Labs)

- Correct the flaws in the Kurkowski information visualization framework, most notably the lack of flexibility

All these goals have been achieved, to one extent or another, as described in the following paragraphs.

5.1.1 Support of Decomposition of Functionality

Due to its object-oriented nature, Java lends itself inherently to the ability to decompose functionality. The ColVis framework further supports decomposition of functionality

with separately designed visualization modules that can be installed in various combinations within the framework.

5.1.2 Facilitating Reuse in Future Software Development

The ColVis framework is particularly suited to code reuse. The underlying framework code, which handles the network functionality, among other things, would be a part of any application based upon it, naturally. In addition, several toolkit classes were developed, comprising some common collaborative/visualization functionality. Most notable amongst these were a whiteboard and snapshot history module, both of which could be added/removed from the application at runtime through the use of a control center module. Also, any visualization modules developed in the framework could be reused in future applications as-is by simply plugging them in alongside other modules within the framework, if appropriate.

5.1.3 Development of Components Improving Existing Analysis Techniques

In order to improve the ability to analyze the output of a chemical plume simulation, a custom module was designed within the framework to allow collaborative control and analysis of the simulation. It allows any number of users to be able to set key input parameters to the simulation, then view and annotate a visualization of the results. Secondary modules allow for additional manipulation of the visualization as described in the fourth chapter. As a result, geographically separated collaborators could lend their respective expertise to better analyze the simulation data as a group.

5.1.4 Allow for Adequate Flexibility

Flexibility was a key concern in the design of the ColVis framework. The framework imposes no real restrictions on a visualization module developed within it, beyond representing the module as a tab within a Java Swing JTabbedPane. A visualization module is an implementation of a Swing JPanel, a fundamental and general-purpose GUI component. The visualization module can send any messages to the network, and handle incoming messages any way it needs to. The messages can contain any Java objects as parameters, as long as they are serializable, which is a requirement to be able to write them into the JavaSpace. The cost of this flexibility is that little code exists in the framework specifically to aid in the visualization aspect. However, this cost is mitigated by the presence of a toolkit containing reusable modules implementing basic visualization techniques, as well as collaborative components.

5.2 Secondary Goals Achieved

In addition to creating a viable JavaSpaces collaborative visualization framework, this research attempted to devise a means of reducing the bandwidth required by network traffic. To accomplish this, an abstract data wrapper class was developed that allowed a user to easily read in a data file and compress it using Java's built in compression libraries. This method achieved compression ratios as high as 10:1 in practice, a significant improvement.

This research also aimed to demonstrate the viability of the JavaSpaces API as a primary network communication architecture. The performance was shown to be more than adequate in rudimentary, limited-scale testing.

5.3 Summary

The ColVis framework is a simple, elegant, and flexible means for rapidly developing collaborative visualization applications using the JavaSpaces API. Along with several toolkit classes to allow one to quickly implement a non-trivial application, a custom module was designed within the framework to demonstrate its functionality. The product has met all goals established for it and exceeded sponsor expectations, so can safely be considered a success.

References

[1] Beca, Lucasz, et al. "Component Architecture for Building Web-based Synchronous Collaboration Systems".

[2] Burridge, Rich. Java™ Shared Data Toolkit User Guide. Sun Microsystems. 1999.

[3] Card, Stuart K., et al. Readings in Information Visualization: Using Vision to Think Morgan Kaufmann. San Francisco. 1999.

[4] Chabert, Annie, et al. "NCSA Habanero® - Synchronous collaborative framework and environment".

[5] Edwards, W. Keith. Core Jini™. Prentice Hall. Upper Saddle River. 1999.

[6] Fayad, Mohamed E., et al. Building Application Frameworks. Wiley. New York. 1999.

[7] Freeman, Eric, et al. JavaSpaces™: Principles, Patterns, and Practice. Addison-Wesley. Reading. 1999.

[8] Inselberg, Alfred. "Multidimensional Detective".

[9] "Jini™ Network Technology: Overview". http://www.sun.com/jini/overview

[10] "Joint Battlespace Infosphere (JBI)". http://www.rl.af.mil/programs/jbi/

[11] Kurkowski, Stuart. "An Information Visualization Solution for the Analysis of AFM Simulation Output Data". Air Force Institute of Technology, 2000.

[12] Schrage, Michael. Shared Minds. Random House. New York. 1990.

[13] Shneiderman, Ben. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley. Reading. 1998.

[14] Wood, J. "Collaborative Visualization". University of Leeds, School of Computer Studies. 1998.

Vita

2Lt Sean C. Butler was born   in Fort Ord, California. He graduated from West Forsyth High School in May 1996. He completed his undergraduate studies at the University of Southern California in Los Angeles, California where he graduated with a Bachelor of Science degree in Computer Engineering and was commissioned in May 1999 through ROTC Detachment 060.

In August 1999, he entered into the Computer Engineering Masters program, Graduate School of Engineering and Management, Air Force Institute of Technology, his first assignment. Upon graduation, he will be assigned to the Air Force Information Warfare Center (AFIWC) at Kelly AFB, San Antonio, TX.

1. REPORT DATE *(DD-MM-YYYY)* xx-03-2001	2. REPORT TYPE Master's Thesis	3. DATES COVERED Mar 2000-Mar 2001

4. TITLE AND SUBTITLE A FLEXIBLE FRAMEWORK FOR COLLABORATIVE VISUALIZATION APPLICATIONS USING JAVASPACES	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Butler, Sean C., Second Lieutenant, USAF	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCE/ENG/01M-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Douglas Holzhauer Air Force Research Laboratory/IFTC 26 Electronics Parkway Rome, NY 13441-4514 Phone: (315) 330-4920 Email: djh@rl.af.mil	10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/IFTC
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES
Lt. Col. Timothy M. Jacobs, ENG, (937) 255-6565 x4279, Timothy.Jacobs@afit.edu

14. ABSTRACT

The complexity of modern tasks is rising along with the level of technology. Two techniques commonly used to deal with complexity are collaboration and information visualization. Recently, computer networks have arisen as a powerful means of collaboration, and many new technologies are being developed to better utilize them. Among the newer, more promising of these technologies is Sun Microsystems' JavaSpaces™, a high-level network programming API. This thesis describes a tool for developing collaborative visualization software using JavaSpaces-an application framework and accompanying toolkit.

In addition to a detailed description of the framework, the thesis also describes an application implemented using the framework, discusses the benefits of development under the framework, evaluates the performance of JavaSpaces in the context of the framework, and addresses the issue of network bandwidth limitations, which are a concern when developing visualizations that deal with large data sets.

15. SUBJECT TERMS
Collaboration, Visualization, Application Framework, JavaSpaces

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Lt. Col. Timothy M. Jacobs
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	58	(937) 255-6565 x4279

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18